Appendix D. Elevator Control System Case Study

This appendix describes an application of the proof-of-concept prototype, CODA, described in Chapter 10, to an elevator control system. The specification for this system consists of a set of hierarchically arranged data/control flow diagrams, one state-transition diagram, and a textual description. The specification is taken from Gomaa.[1] [Gomaa93, Chapter 24] From this specification, CODA generates, with the assistance of an experienced designer, four designs. One design, for an elevator system that has two elevators and five floors, uses the default target environment description. For each of two additional designs, the designer reuses much of the first design but assigns different priorities for some of the messages exchanged between tasks. In one case, however, the target environment does not provide priority, message-queuing services, while in the second case the target environment does provide priority, message-queuing services. These two designs demonstrate how CODA adjusts a design to map priority messages onto the available services of an intended target environment. The fourth design, which assumes the default target environment description, considers an elevator system for a large building, containing 12 elevators and 48 floors.

---

[1] The elevator control problem dates back to at least Donald Knuth. The interested reader can consult a number of treatments of this real-time application. [Armstrong93, Jackson83, Knuth68, Meyer93, Sanden94]

D.1 Elevator Control System for a Small Building

Figure 59 shows the context diagram for Gomaa's elevator control system. As in previous case studies, the context diagram is annotated with information inferred or elicited when CODA analyzes the specification. The context diagram differs from Gomaa's in only one way. Events arriving from external devices are shown in Figure 59 as dashed, directed arcs. These events, needed to aid in automatic classification of concepts, are not shown in Gomaa's context diagram, but can be inferred to exist from reading the accompanying textual specification.

D.1.1 Analyzing the Specification

Once the data/control flow diagram hierarchy is flattened, the entire data/control flow diagram for the elevator control system consists of 23 nodes (14 transformations, 8 terminators, and 1 data store) and 44 arcs (23 data flows and 21 event flows). After loading the specification, the designer asks CODA to analyze the specification. CODA begins by attempting to classify concepts on the data/control flow diagram.

D.1.1.1 Classifying Concepts

CODA inquires about the nature of the terminators and then proceeds with concept classification until, during stimulus-response classification, an ambiguity appears. CODA cannot determine which, if either, of a pair of data flows exchanged between two transformations, Scheduler and Accept New Request, is sent in response to the other. On this point, CODA consults the designer. The designer recognizes that neither of these data flows is sent in response to the other; rather the data flows are

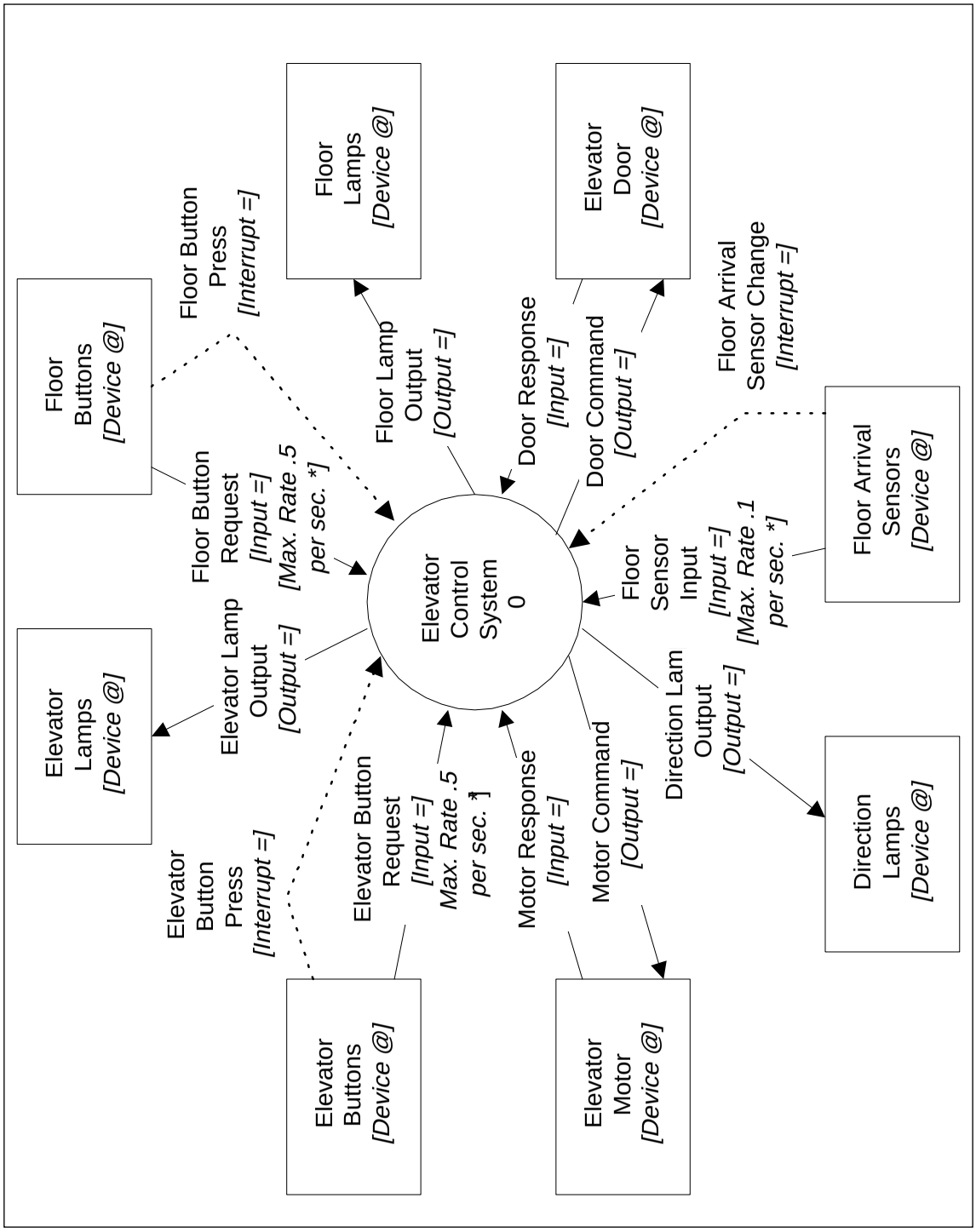

Figure 59. Context Diagram for an Elevator Control System

independent of each other.

Another ambiguity arises during the final stage of classification. CODA cannot establish whether one transformation, Check This Floor, is a synchronous function or an asynchronous function. Upon consulting the designer, CODA learns that the time spent executing the transformation might unduly delay the invoking function, Floor Arrival Sensor (see Figures 62 and 63, where the relevant elements of the data/control flow diagram are shown). The designer made this judgment after reading the textual specification accompanying the data/control flow diagrams. CODA completes the remainder of the concept classifications without consulting the designer.

D.1.1.2 Eliciting Additional Information and Verifying the Specification

After all concepts are classified, CODA determines that certain input data flows require a maximum rate. This information can be used during design evaluation to help assess the schedulability of the design. Next, finding no exclusion groups defined for the specification, CODA invites the designer to define one or more exclusion groups. The designer declines. CODA then invites the designer to define aggregation groups. Based upon the textual specification accompanying the data/control flow diagrams, the designer recognizes that each elevator comprises a door, a motor, a set of lamps, and a set of buttons. For this reason, the designer accepts CODA's offer and defines an appropriate aggregation group for the elevator.

Next, CODA, finding no locked-state events defined for the specification, offers the designer the chance to denote locked-state events. The designer analyzes the

state-transition diagram, Figure 60, for the only control transformation, Elevator Controller, and finds eight locked-state events. These locked-state events occur because the Elevator Controller, in most instances, takes one action and then awaits a response before moving on to a new state. In fact, only two event flows, Up Request and Down Request, do not qualify as locked-state events. These event flows do not qualify because each of them can arrive any time a client presses a floor button or when the scheduler schedules an elevator. The remaining events can only arrive when the Elevator Controller is expecting them.

After completing elicitation of specification addenda, CODA inquires whether the designer wishes to update the cardinality of any nodes.[2] Here, the designer decides to establish cardinalities for each node in order to correspond to a specific size of building, five floors, and a specific number of elevators, two. Ideally, a design for the elevator control system would be independent of cardinality; however, CODA makes a number of decisions about task inversion and module placement that depend upon the relative cardinalities of various transformations and data stores.

To complete the specification analysis, the designer checks the state of concept classification and also verifies that all axioms are satisfied. The specification is then saved in preparation for design generation.

D.1.1.3 Annotated Data/Control Flow Diagram

The results from CODA's analysis of the specification are presented as an annotated data/control flow diagram, shown in Figures 61 through 64. Figure 61

[2] The default node cardinality is one.

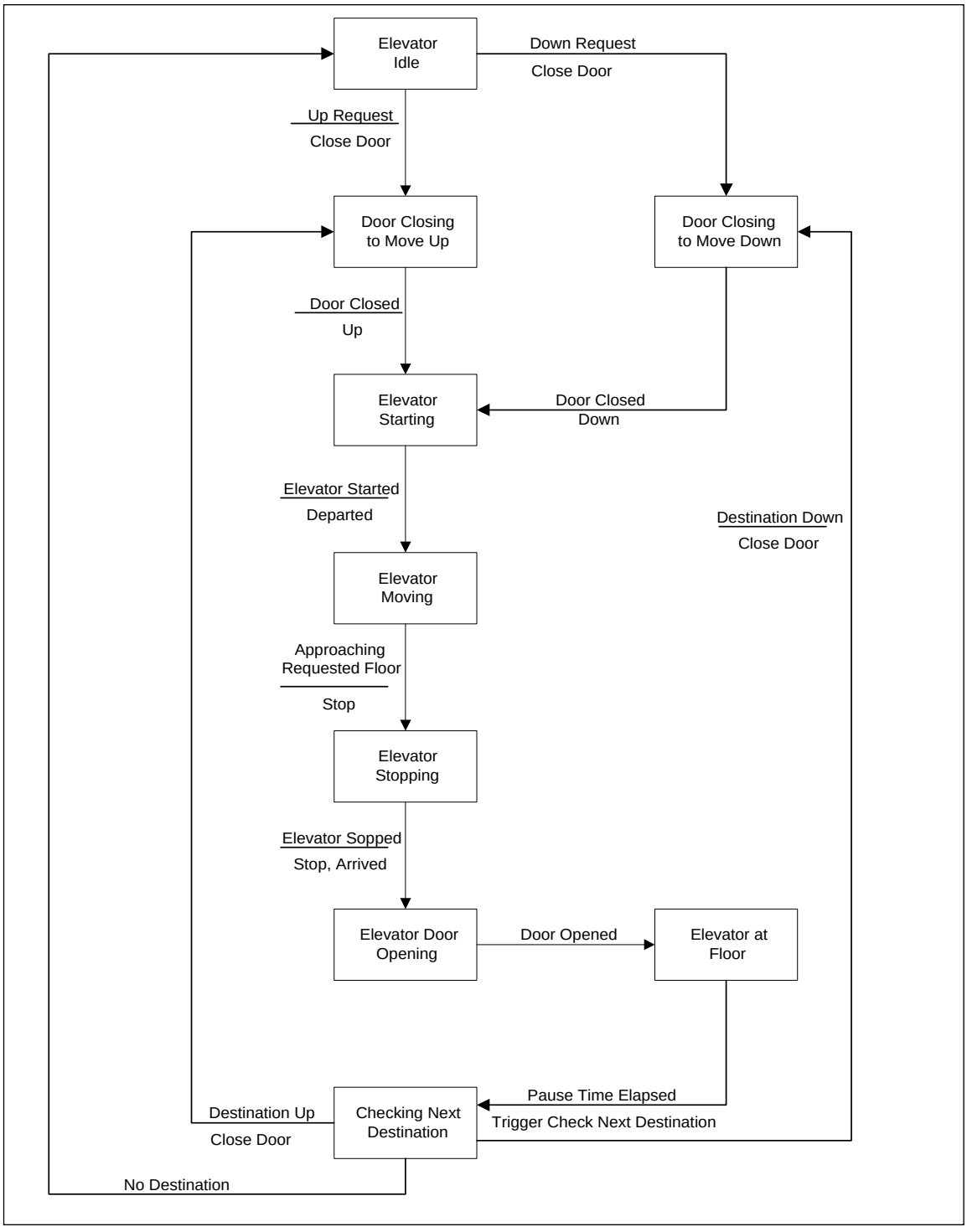

Figure 60. State-Transition Diagram for the Elevator Controller

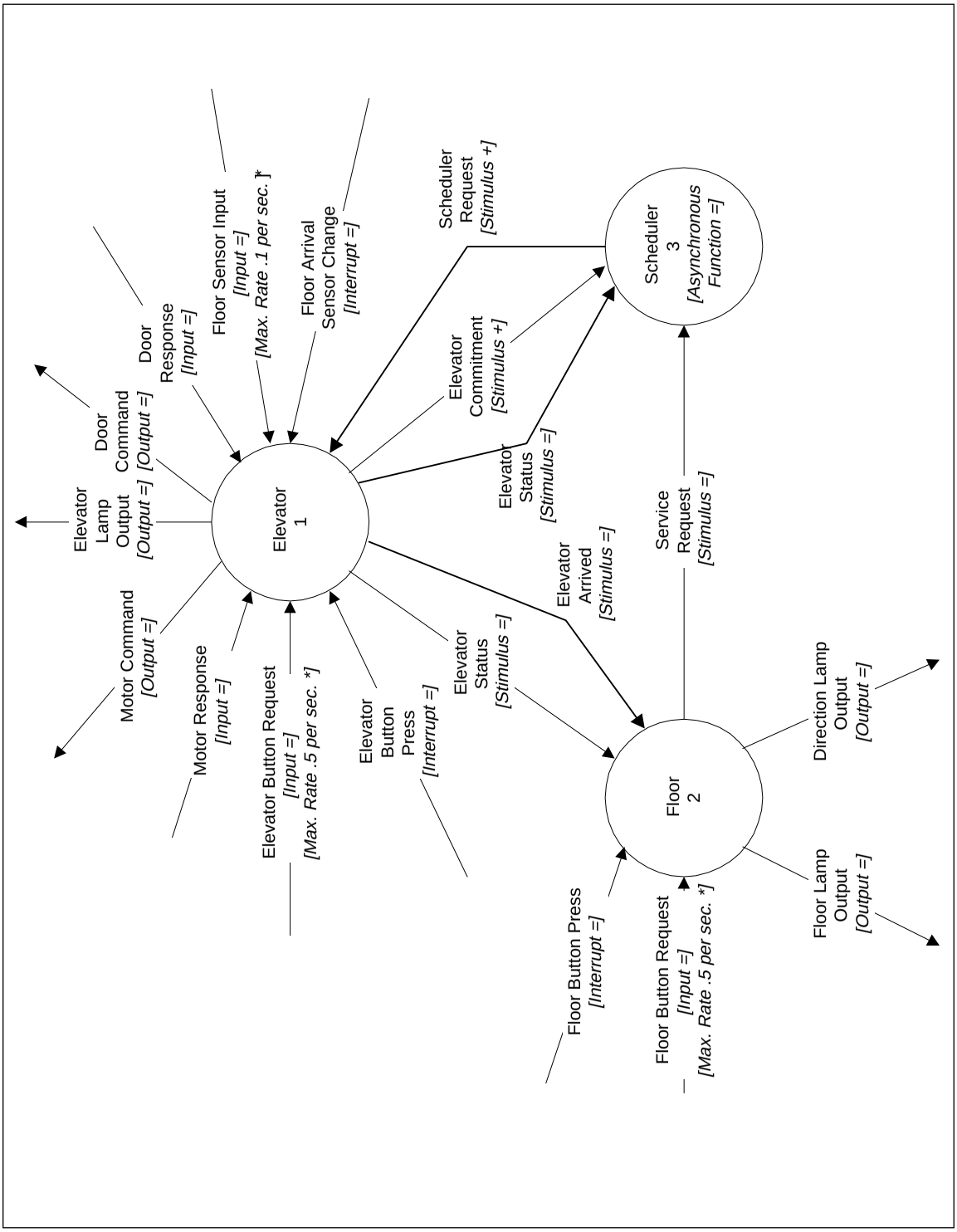

Figure 61. Subsystem Decomposition of the Elevator Control System

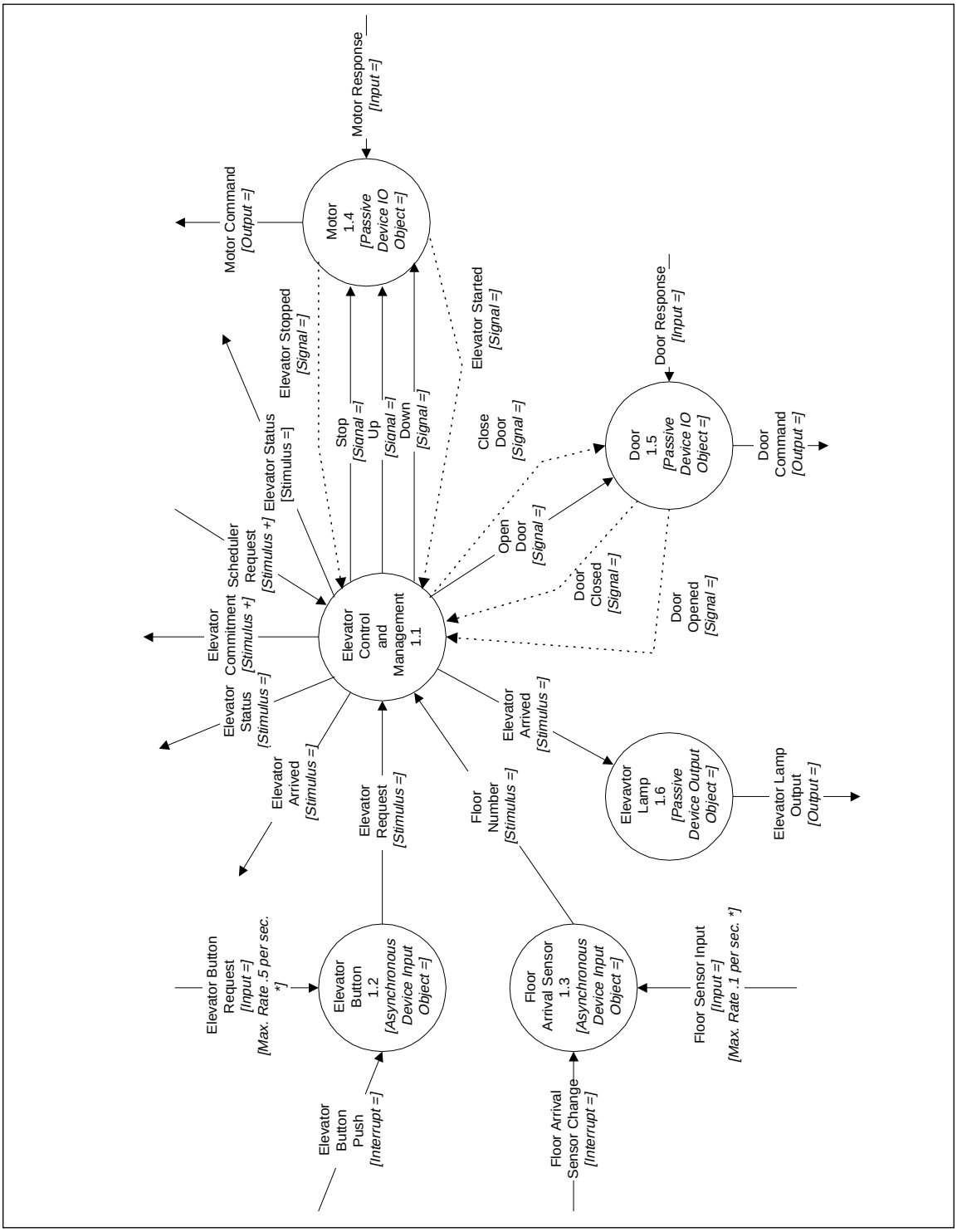

Figure 62. Decomposition of the Elevator Subsystem

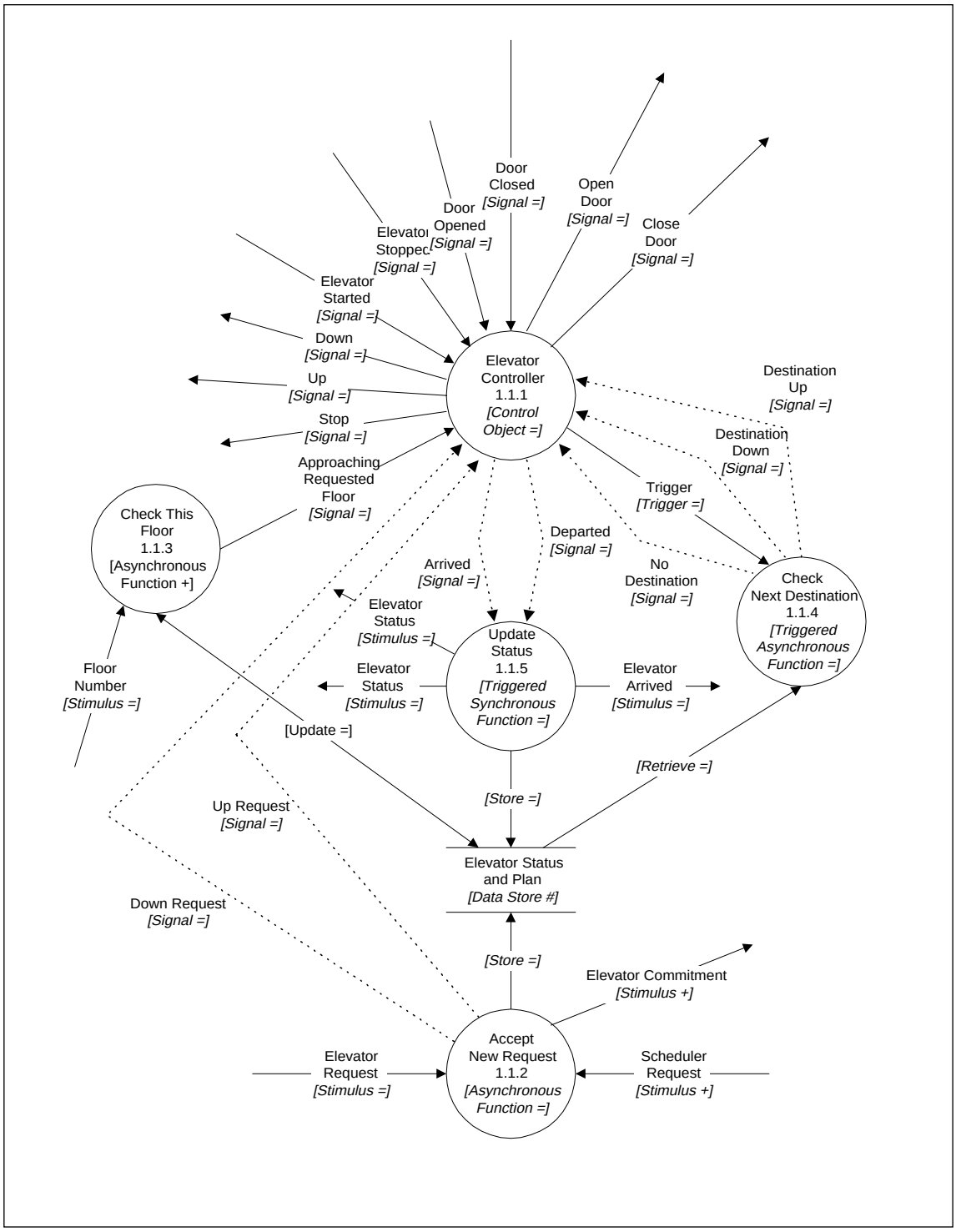

Figure 63. Decomposition of Elevator Control and Management

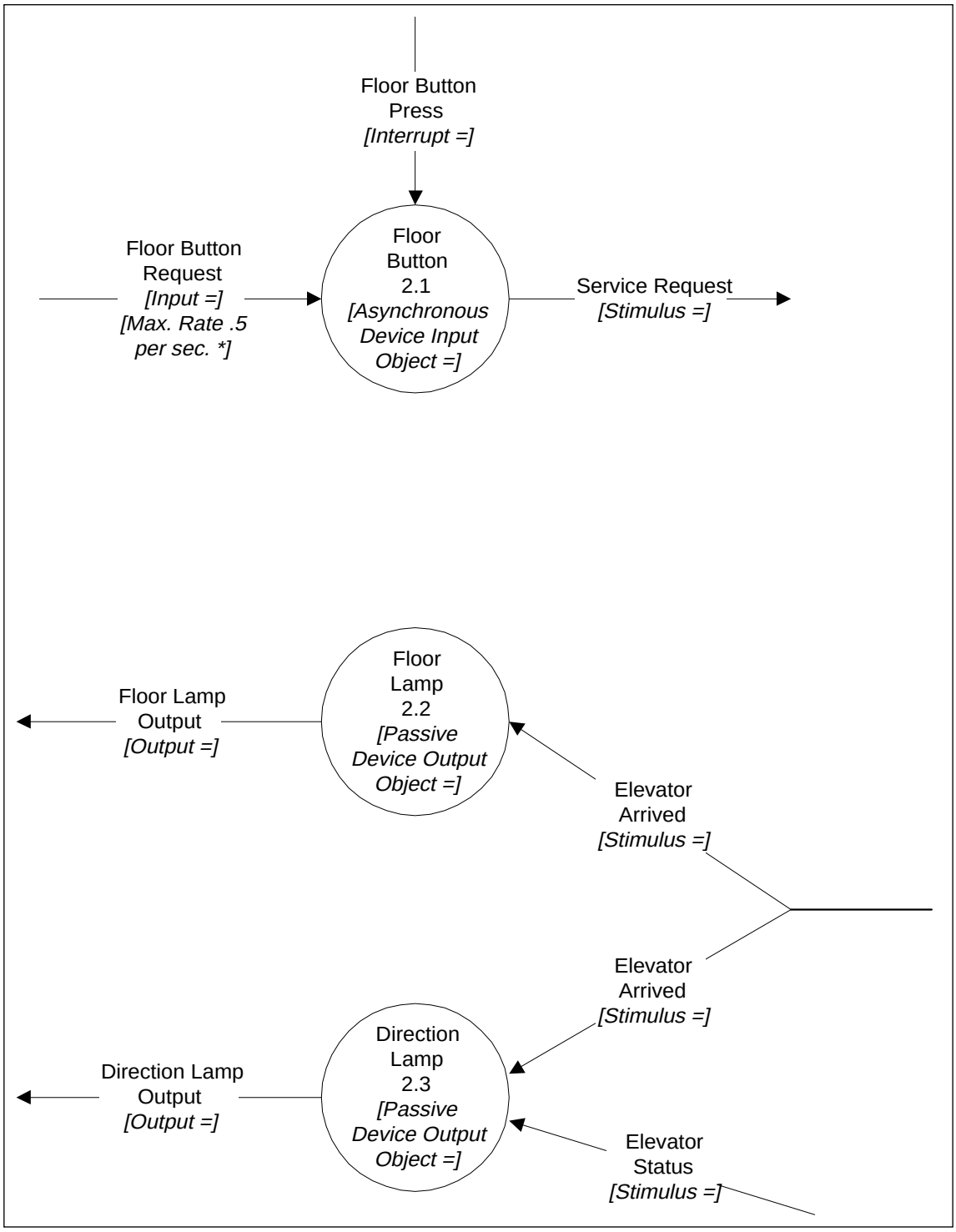

Figure 64. Decomposition of the Floor Subsystem

decomposes the elevator control system into three subsystems. Two of these subsystems, Elevator and Floor, are further decomposed. The decomposition of the Elevator subsystem is given in Figure 62. Five of the six transformations in Figure 62 are annotated with their classifications; one transformation, Elevator Control and Management, is decomposed further, as illustrated in Figure 63. The decomposition of the Floor subsystem is depicted in Figure 64.

D.1.2 Generating the Design

After completing specification analysis, the designer decides to structure tasks as a first step in the process of generating a concurrent design. CODA requires no consultation with the designer, except as needed to assign new task names.

D.1.2.1 Structuring Tasks

Table 47 gives the outcome of CODA's task structuring, including: the tasks created, the transformations allocated to each task, and the criterion used in determining each allocation. The designer saves the results of the task structuring before moving on to other design chores.

D.1.2.2 Structuring Modules

Next, the designer decides to structure the transformations and data stores into information hiding modules. As with the task structuring, CODA requires no consultation with the designer, except for renaming modules and operations. The designer saves the results from module structuring. Table 48 reports the results of CODA's module structuring for the elevator control system.

Table 47. Task Structuring Decisions for Elevator Control System

Task	**Transformation**	**Structuring Criterion**
Elevator Manager	Accept New Request	Asynchronous Internal Task
Scheduler	Scheduler	Asynchronous Internal Task
Monitor Elevator Buttons	Elevator Button	Asynchronous Device I/O Task
Monitor Floor Buttons	Floor Button	Asynchronous Device I/O Task
Monitor Arrival Sensors	Floor Arrival Sensor	Asynchronous Device I/O Task
Elevator Controller	Elevator Controller Check This Floor Check Next Destination Door Motor Elevator Lamp Update Status	Control Task Asynchronous Internal Task & Sequential Cohesion Triggered Asynchronous Task & Control Cohesion Aggregation Aggregation Aggregation Control Cohesion
Floor Lamps Monitor	Floor Lamp	Resource Monitor Task
Direction Lamps Monitor	Direction Lamp	Resource Monitor Task

Table 48. Module Structuring Decisions for Elevator Control System

Module	**Transformation/Data Store**	**Structuring Criterion**
Elevator Status and Plan	Elevator Status and Plan Check Next Destination Check This Floor Accept New Request Update Status	Data-Abstraction Module Read Operation of DAM Update Operation of DAM Update Operation of DAM Update Operation of DAM
Elevator Control STM	Elevator Controller	State-Transition Module
Elevator Button	Elevator Button	Device-Interface Module
Floor Button	Floor Button	Device-Interface Module
Arrival Sensor	Floor Arrival Sensor	Device-Interface Module
Floor Lamp	Floor Lamp	Device-Interface Module
Direction Lamp	Direction Lamp	Device-Interface Module
Door	Door	Device-Interface Module
Motor	Motor	Device-Interface Module
Elevator Lamp	Elevator Lamp	Device-Interface Module
Scheduler Algorithm	Scheduler	Algorithm-Hiding Module

D.1.2.3 Integrating Tasks and Modules

Once both tasks and modules are structured, the designer decides to integrate the two views. CODA completes the integration without help from the designer.

D.1.2.4 Defining Task Interfaces

To complete the design for the elevator control system, the designer needs only to define the interfaces among tasks. CODA does not need to consult with the designer for most decisions; however, since the designer is experienced, CODA, noticing that at least one task receives queued messages from multiple sources, invites the designer to consider assigning different priorities to the appropriate messages. In this case, the designer declines the invitation.

After the task interfaces are defined, CODA offers to let the designer review and rename the new design elements. Once renaming is completed, the designer saves the design and then asks that task and module specifications and design histories be written to disk. Following the generation of specifications and design histories, CODA checks the design for completeness and consistency and writes a design summary to disk. Figures 65 and 66 show, in two parts, the design summary and completeness and consistency report for the elevator control system design.

D.1.2.5 The Completed Design

The design generated by CODA for the elevator control system is depicted in Figure 67. This design differs in several details from the solution given by Gomaa. One difference of note is that CODA's design includes two Elevator Manager tasks, while

Specification: Elevator Control System

TED: A Default Environment (Message Queues, No Priority Queues)

8 TASKS

TASK 1 Elevator_Manager
TASK 2 Scheduler
TASK 3 Monitor_Elevator_Buttons
TASK 4 Monitor_Floor_Buttons
TASK 5 Monitor_Arrival_Sensors
TASK 6 Elevator_Controller
TASK 7 Floor_Lamps_Monitor
TASK 8 Direction_Lamps_Monitor

11 MODULES

IHM 1 Elevator_Status_and_Plan
IHM 2 Elevator_Control_STM
IHM 3 Elevator_Button
IHM 4 Floor_Button
IHM 5 Arrival_Sensor
IHM 6 Floor_Lamp
IHM 7 Direction_Lamp
IHM 8 Door
IHM 9 Motor
IHM 10 Elevator_Lamp
IHM 11 Scheduler_Algorithm

COMPLETENESS REPORT

Each transformation is allocated to at least one task

Each transformation is allocated to a module

Each data store is allocated to a module

Each directed arc is allocated to the design

Each two-way arc is allocated to the design

Figure 65. Design Summary and Completeness and Consistency Report
(Part One of Two)

```
CONSISTENCY REPORT
------------------

Each module is either contained in a task or
  is accessed by a task or by another module
Each module provides at least one operation
Each operation is provided by a module
Each task receives at least one input
Each task appears to write some output
Each internal event is both generated by and accepted by a task
Each external event and each timer event is accepted by a task
Each datum is either read or written by some task
Each message is either sent and received by a task
    or is carried in another message
Each queued message is held by a queue
Each queue holds at least one message
Each queue is either consumed by or encapsulated within a task
  or is a subqueue for a priority queue
Each priority queue heads at least one subqueue
Each priority queue is either owned by or enclosed within a task
Each message data is included in a message
Each parameter is either taken, yielded, or altered by some operation
No operation is provided by more than one module
No module is contained by more than one task
No event is accepted by more than one task
No event is generated by more than one task
No datum is written by more than one task
No queue is consumed by more than one task
No queue is encapsulated by more than one task
No queue is both consumed and encapsulated
No priority queue is owned by more than one task
No priority queue is enclosed by more than one task
No priority queue is both owned and enclosed
No queue is a subqueue more than one priority queue
No queued message is held by more than one queue
No message is sent by more than one task
No message is received by more than one task
No message datum is included in more than one message
Each message carried in another message is carried in exactly two messages
No tightly-coupled message answers more than one message
No parameter is yielded by more than one operation
No parameter is taken by more than one operation
No parameter is altered by more than one operation
No parameter is both yielded and altered
```

Figure 66. Design Summary and Completeness and Consistency Report
(Part Two of Two)

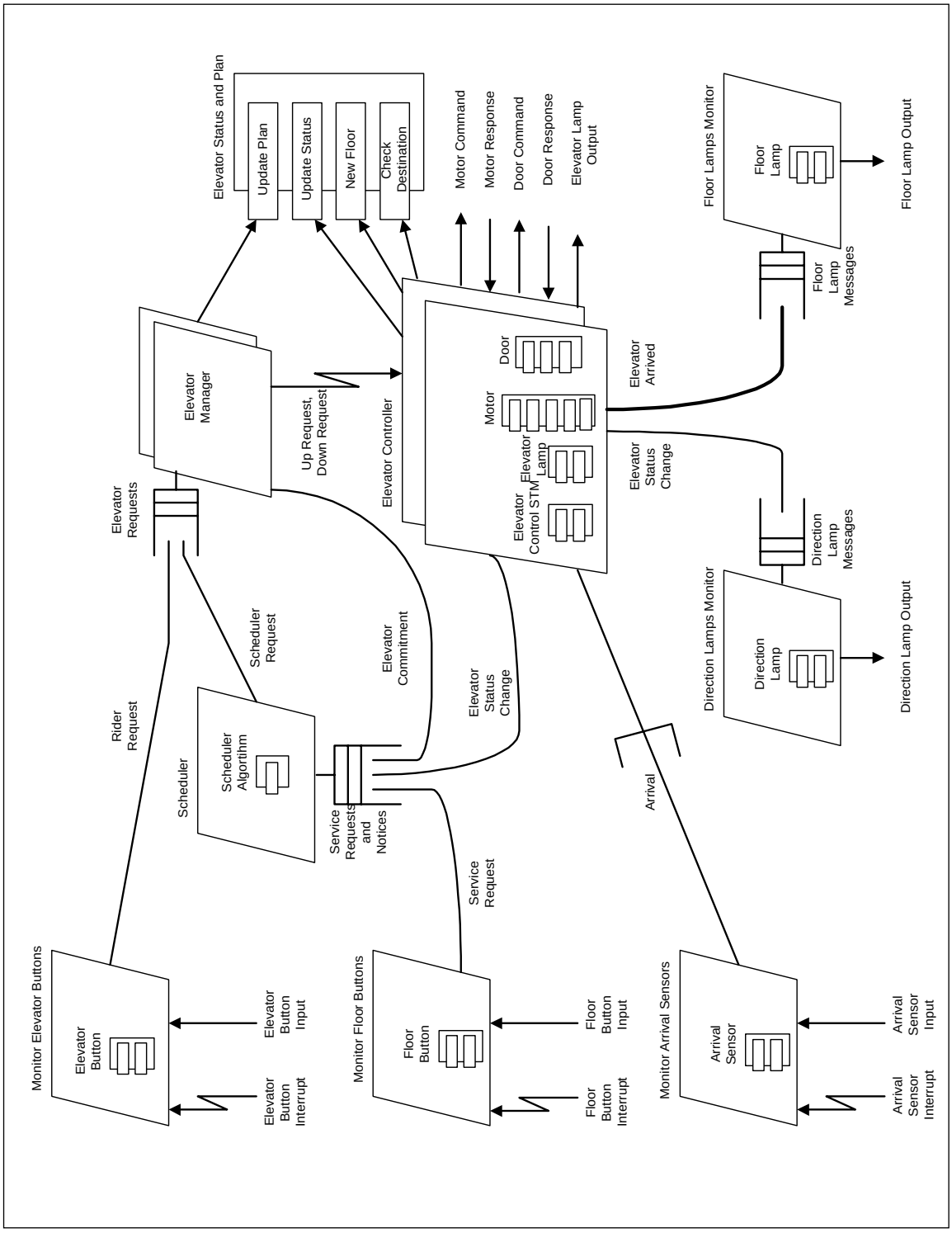

Figure 67. Generated Design for the Elevator Control System - Default Target Environment - No Messages Assigned Priority

Gomaa's solution uses a single Elevator Manager task. CODA produces two tasks based upon the cardinality of the transformation, Accept New Request, from which the Elevator Manager task is allocated. Gomaa decided that only one Elevator Manager task is required. Gomaa's decision results from an understanding that the processing in the transformation, Accept New Request, is rather simple and easy to invert, whereas, the processing represented by the control transformation, Elevator Controller, and its aggregated objects, is complicated and more difficult to invert. In addition, Gomaa envisions a design where a single Elevator Manager task can provide inputs to numerous Elevator Controller tasks. CODA is incapable of reaching this same conclusion from the facts at hand.

A second difference occurs in the interface between the Elevator Manager and Elevator Controller tasks. CODA maps two event flows, Up Request and Down Request, from the Elevator Manager onto software signals for the Elevator Controller task. CODA takes this mapping because the Elevator Manager and the Elevator Controller must synchronize on these events and because the default target environment description indicates that up to two inter-task signals are supported between each pair of tasks in a design. Gomaa chooses to map these event flows onto a tightly-coupled message. CODA might also choose to map these event flows to a tightly-coupled message, but only when the number of event flows to be sent between a pair of tasks exceeds the maximum permitted for the target environment.

A third difference between CODA's design and Gomaa's solution involves the interface between the Scheduler and two other tasks: the Elevator Manager and the Elevator Controller. CODA maps two data flows, Elevator Commitment from the Elevator Manager and Elevator Status from the Elevator Controller, to queued messages for the Scheduler. Gomaa, instead, maps these data flows onto accesses to a data store, Elevator Status and Plan. This difference results from the fact that Gomaa constructed the data/control flow diagram in anticipation of a distributed solution. Examined in light of distributing the Scheduler task to a separate processor from the Elevator Manager and Elevator Controller tasks, the data flows between these tasks must be mapped to messages. When Gomaa, instead, uses the same data/control flow diagram to generate a design for a single-processor system, he decides simply to replace the data flows to the Scheduler with accesses to a data store, Elevator Status and Plan, shared among all three tasks. As a secondary effect, Gomaa's solution shows an operation, Select Elevator, for a data-abstraction module, Elevator Status and Plan, not included in the design generated by CODA. Gomaa's alternative mapping leads to an efficient solution for cases where the design will run an a single processor system. CODA is unable to make these kinds of adjustments, and must, therefore, adhere more closely to the structures shown on the data/control flow diagram.

The mapping generated by CODA, while suitable for a distributed design, leads to an inefficient solution for cases where the design will execute on a single processor. For CODA's design, the Scheduler must be assumed to maintain an independent view of the

plan and status for all elevators and must be assumed to update this view based upon messages received from the Elevator Manager and Elevator Controller tasks. Since the Scheduler subsystem is not decomposed further on the data/control flow diagram, these additional details are not included in the design.

Another difference between the design generated by CODA and the solution provided by Gomaa involves an operation of the Elevator Status and Plan data-abstraction module. CODA generates an operation, Update Status, not included in Gomaa's solution. In a discussion about this point, Gomaa indicated to the author that the operation should be included in his solution. The operation was simply overlooked in the design produced by Gomaa. [Gomaa93, Chapter24]

D.1.3 Differentiating Queued Messages by Priority

Suppose the designer believes that the elevator control system will run more smoothly if preference is given to decisions that involve the scheduler. If this were the case, the designer might decide to give higher priority to messages exchanged between the Scheduler task and the internal, control-related tasks, that is, the Elevator Manager and the Elevator Controller tasks. This policy can be incorporated into the design by redefining the task interfaces. First, the designer moves an appropriate copy of the design produced previously, that is, a design with the task and module structures defined and integrated, to a new workspace. The movement of designs between workspaces is conducted outside the prototype, using file copy facilities from the hosting operating system. Then, the designer restores the partial design and repeats the definition of the

task interfaces. Since the designer is experienced, CODA invites the designer to assign message priorities. In this case, the designer accepts the invitation and assigns a higher priority to requests received by the Elevator Manager task from the Scheduler task. The designer also assigns higher priority to messages received by the Scheduler when those messages arrive from the Elevator Manager task or the Elevator Controller task. In effect, messages generated within the elevator control system are given preference over requests from outside the system. After message priorities are assigned, CODA allocates message-queuing mechanisms, as appropriate, and then invites the designer to review and rename task-interface elements. Once renaming is complete, the designer saves the design and then writes the specifications, histories, and summary to disk.

D.1.3.1 Priority Message Queuing Services Unavailable

The alternate design generated by CODA based on varying message priorities is shown in Figure 68. Since the designer uses the description for the default target environment, priority message queues are unavailable. For this reason, CODA simulates priority message queues by allocating a separate message queue for each priority level and then maps messages into the appropriate queues by priority and destination.

D.1.3.2 Priority Message Queuing Services Available

CODA need not have simulated priority message queues had the designer chosen to use a target environment description where priority message queues are available. Suppose the designer loads a target environment description that includes priority message queues. Loading a new target environment description requires any design in

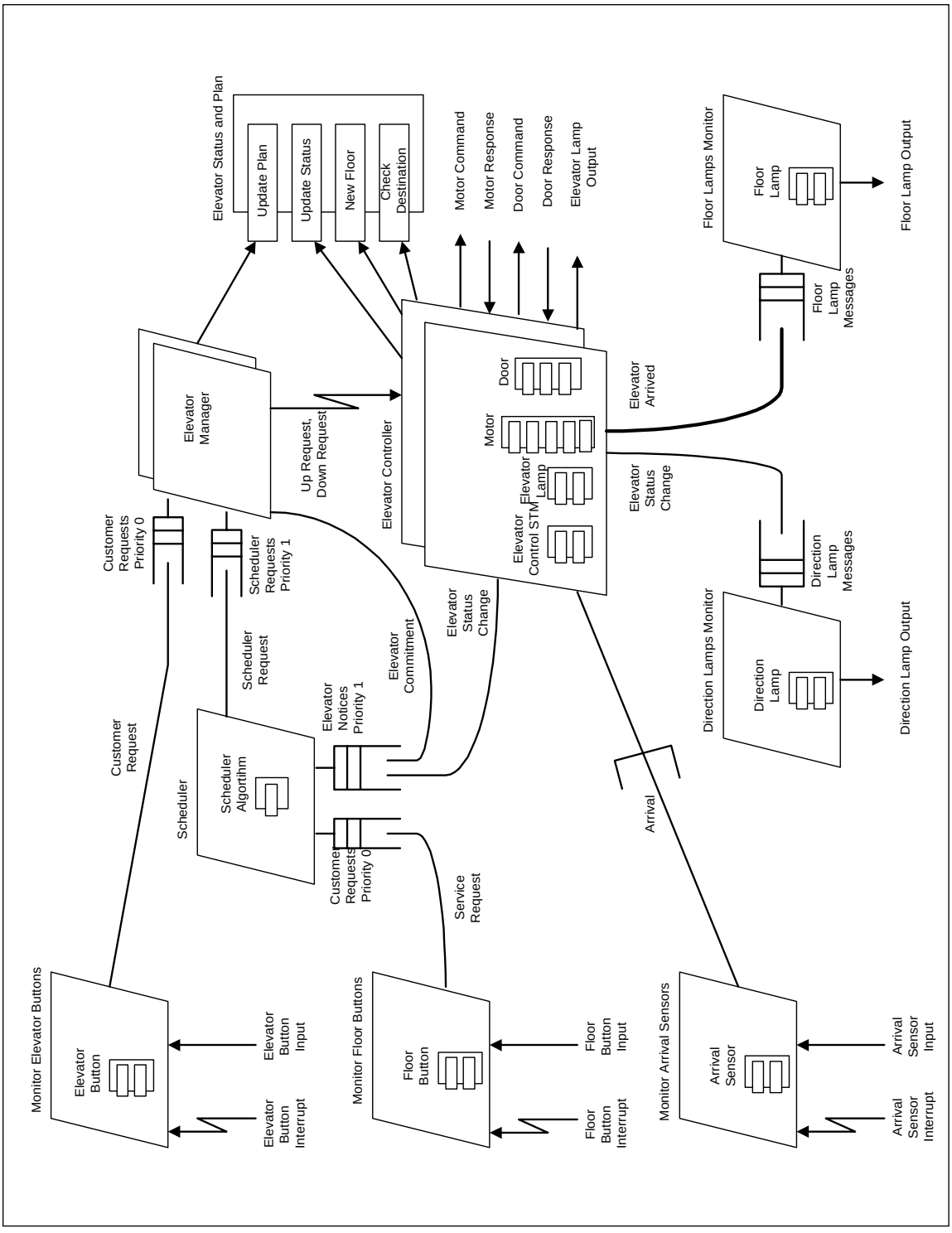

Figure 68. Design for the Elevator Control System - Default Target Environment - Queued Messages Assigned Varying Priorities

progress to be deleted from memory because the existing design might no longer reflect the proper target environment. The designer then restores an existing copy of a partially completed design that was moved previously to the appropriate workspace. Assuming that the preexisting design lacks a definition for task interfaces and that the designer asks CODA to define task interfaces, Figure 69 shows the alternate design generated based on the new target environment description.

D.2 Elevator Control System for a Large Building

Suppose that a design is required to control twelve elevators in a large building with 48 floors. Given the description for the default target environment, CODA extends its use of task inversion to cover the Elevator Manager and Elevator Controller tasks. This occurs because the total of twelve elevators exceeds the task inversion threshold, set at eight for the default target environment. The resulting inversion of the Elevator Manager and Elevator Controller tasks causes CODA to define some of the interfaces for these tasks differently from the interfaces defined in the previous designs for a smaller building. Figure 70 depicts the design for a large building, as generated by the prototype.

The Elevator Manager and the Elevator Controller tasks, replicated in previous designs, are now each implemented by a single task. Context switching between elevators must be handled within each of the tasks. Consequently, the Direction Lamp and Floor Lamp modules are no longer accessed by multiple tasks. For this reason, these two, device-interface modules are now placed within the Elevator Controller task, rather than within separate, resource-monitor tasks. Two other design changes also appear.

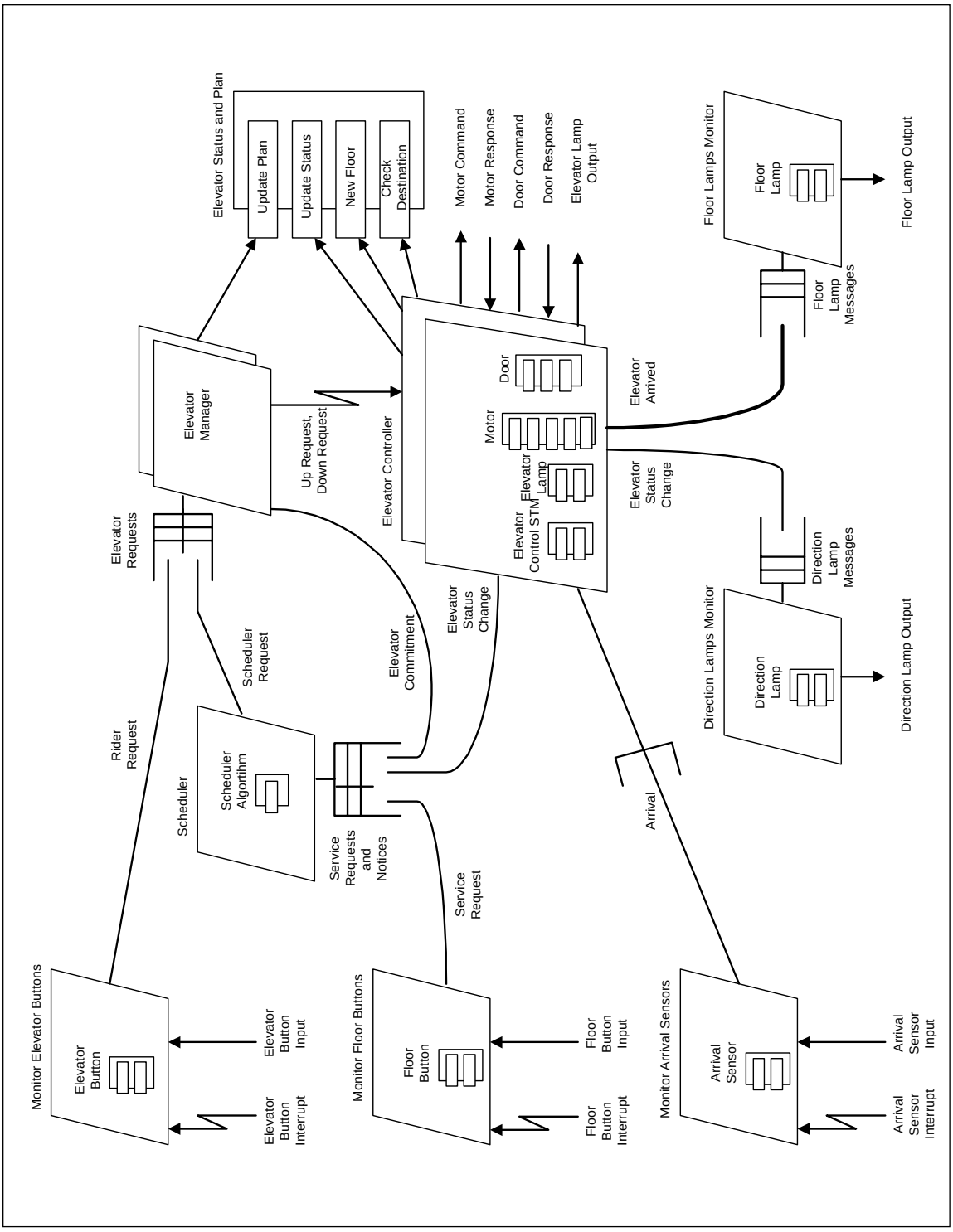

Figure 69. Design for the Elevator Control System - Priority Message Queues Available - Queued Messages Assigned Varying Priorities

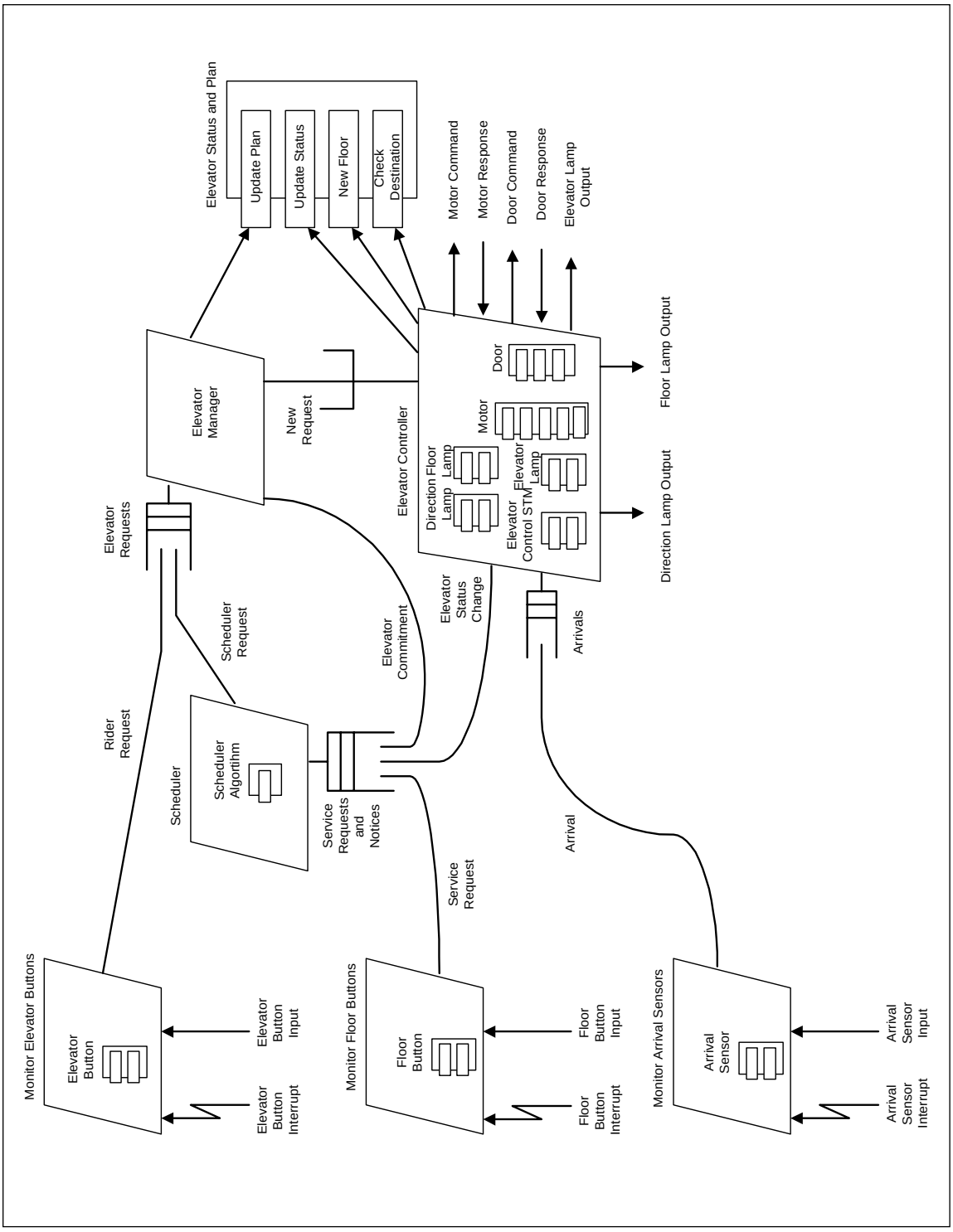

Figure 70. Design for the Elevator Control System - Default Target Environment - Large Building (12 Elevators and 48 Floors)

First, the interface between the Elevator Manager and the Elevator Controller tasks is now defined to be a tightly-coupled message, rather than two software signals. CODA makes this change because the destination of the event flows, Up Request and Down Request, can no longer be identified solely from the receiving task. Instead, the Elevator Manager task must include data along with each event flow to identify the specific elevator for which the events apply. A second design change involves the Arrival messages received by the Elevator Controller task from the Monitor Arrival Sensors task. These messages are now placed into a message queue, Arrivals, for the receiving task, Elevator Controller. Since the Elevator Controller task handles multiple elevators, the possibility exists that the Elevator Controller task will not be ready to receive an Arrival message at the time the message is sent. To avoid delaying the Monitor Arrival Sensors task, the Arrival messages must be queued.

Several differences in the various designs generated by CODA for the elevator control system result from differences in element cardinalities, relative to the task inversion threshold. Ideally, a designer might strive to produce a design that does not vary based on the number of elevators and floors within the target building. To achieve this goal, a design for a distributed elevator control system appears attractive. Such a design requires replicating the Elevator Controller task and the Elevator Manager task for each elevator and mapping each pair of these tasks to a processor for each elevator. The tasks for the floor subsystem might then be allocated to a separate processor, as could the Scheduler task. As currently defined, CODA cannot contemplate such trade-offs.